JMLR: Workshop and Conference Proceedings 21:243–248, 2012 The 11th ICGI

# Results of the PAutomaC Probabilistic Automaton Learning Competition*

**Sicco Verwer**† s.verwer@cs.ru.nl
*Radboud University Nijmegen*

**Rémi Eyraud** remi.eyraud@lif.univ-mrs.fr
*QARMA team, Laboratoire d'Informatique Fondamentale de Marseille*

**Colin de la Higuera** cdlh@univ-nantes.fr
*TALN team, Laboratoire d'Informatique de Nantes Atlantique, Nantes University*

**Editors:** Jeffrey Heinz, Colin de la Higuera, and Tim Oates

## Abstract

Approximating distributions over strings is a hard learning problem. Typical GI techniques involve using finite state machines as models and attempting to learn both the structure and the weights, simultaneously. The PAutomaC competition is the first challenge to allow comparison between methods and algorithms and builds a first state of the art for these techniques. Both artificial data and real data were proposed and contestants were to try to estimate the probabilities of test strings. The purpose of this paper is to provide an overview of the implementation details of PAutomaC and to report the final results of the competition.

## 1. Introduction

The PAutomaC probabilistic automaton learning competition was an on-line challenge that can be found at http://ai.cs.umbc.edu/icgi2012/challenge/Pautomac/. The goal of PAutomaC was to provide an overview of which probabilistic automaton learning techniques work best in which setting and to stimulate the development of new techniques for learning distributions over strings. Many probabilistic automata learning methods have been produced in the past (see Verwer et al. (2012) for an overview). Due to the difficulty of the learning problem, most of them focus on learning some form of deterministic probabilistic automaton (Dpfa, see, e.g., Carrasco and Oncina (1994)), where only the symbols are drawn from probability distributions but the transitions are uniquely determined given the generated symbol. There exist some exceptions, however, which aim to learn hidden Markov models (Hmm) (Baum, 1972), probabilistic residual automata (Esposito et al., 2002), and multiplicity automata (Denis et al., 2006). Another important approach is to learn Markov chains or $n$-grams by simply counting the occurrences of substrings. These simple counting methods have been very successful in practice (Brill et al., 1998), but there has been

* This work was supported in part by the IST Program of the European Community, under the Pascal Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.
† The first author is supported by STW project 11763 Integrating Testing And Learning of Interface Automata (ITALIA).

so far no thorough investigation of which model/algorithm is likely to perform best and why. PAutomaC aimed to fill this knowledge gap by testing different methods on a set of distributions generated by different types of models of ranging size and sparsity. This is not only very useful for practical applications (where many different types of distributions can be encountered), but also aims to answer to the question whether it is best to learn a non-deterministic model (Hmm, Pfa) or a deterministic model (Dpfa) when the data is drawn from a (non-)deterministic distribution, see, e.g., Gavaldà et al. (2006).

This paper provides a brief overview of PAutomaC and of its results. The participants were given access to train and test sets of these learning problems and asked to learn a string distribution, which should then subsequently be used to assign probabilities of the strings in the test set. In contrast to the traditional method of testing predictive performance, this setup also evaluated whether low probability events were assigned low probabilities. Furthermore, since the learned model structure was not evaluated, it allowed the use of any learning method, not only those resulting in probabilistic automata models. Using train and test sets for performance evaluation also introduced some issues with respect to the evaluation measure and possible collusion, which had been resolved with the help of the PAutomaC scientific committee, see the website for details.

## 2. An overview of PAutomaC

**Generating artificial data.** Artificial data was generated by building a random probabilistic automaton of with 5 to 75 states and with an alphabet consisting of 4 to 24 symbols (both inclusive, and decided uniformly at random). This machine was subsequently used to generate data sets. Of all possible state-symbol pairs that could occur in transitions, between 20 and 80 percent (the symbol sparsity) of them were generated. These pairs were selected by first choosing a state at random, and subsequently choosing a symbol from the set of symbols that had not yet been selected for that state. This created a selection without replacement from the set of all possible state-symbol pairs that was modified to remain uniform over the states. This modification made it less likely that the resulting symbols were evenly distributed over the states. For every generated state-symbol pair, one transition was generated to a randomly chosen target state. Between 0 and 20 percent (the transition sparsity) transitions were generated in addition to these, selected without replacement from the set of possible transitions, modified to remain uniform over the source states and transition labels.

Initial states and final states were selected without replacement until the percentages of selected states exceeded the transition and symbol sparsities, respectively. All initial, symbol, and transition probabilities were drawn from a Dirichlet distribution (making every distribution equally likely). The final probabilities were drawn together with the symbol probabilities. From such a structure, one training and one test set were generated from every target. With probability one out of four, the generated train set was of size 100 000, it was of size 20 000 otherwise. New test strings were generated using the target machine until 1 000 unique strings had been generated. The test strings were allowed to overlap with the strings used for training. If the average length of the generated strings was less than 5 or greater than 50, a new automaton and new data sets were generated using the same construction parameters. In total, 150 models and corresponding train and test sets

were generated using this way. We evaluated the difficulty of the generated sets using a 3-gram baseline algorithm.We then selected 16 of them, aiming to obtain ranging values for the number of states, the size of the alphabet, sparsity values, and difficulty. We applied the same procedure for Dpfa but without generating additional transitions; and for Hmm, we generated state-state pairs instead of state-symbol-state triples.

In total, this results in 48 (16 for every type) artificially generated problems for use in the competition. The participants were given no other information about the target than the two sets of strings.

**Constructing real-world data.** In addition to the artificially generated sets, the competition included two real-world data sets: one from a natural language processing task, and one from time series modeling.

The natural language sets were generated from a corpus consisting of the works of Jules Verne, translated to Dutch. This text was analyzed using the `Frog` Dutch part-of-speech tagger (Van den Bosch et al., 2007). The resulting (Dutch) parts-of-speech were mapped to 11 symbols, and the sentences (separated by dots, commas, or semicolons) were mapped to strings over these symbols. We provided 10 000 of these strings, selected at random, as a train set, and selected 1 000 unique strings as a test set. The performance was evaluated using the scores assigned by the 3-gram baseline, learned on all of the 107 165 created parts-of-speech strings.

The data for the time series modeling task were created using a sliding window of length 20 over a discretized sensor signals that record the fuel usage of a trucks for a Dutch transport company, see Verwer et al. (2011). We provided 20 000 of the resulting sequences of discretized sensor data as a train set, 1 000 unique sequences as a test set, and evaluated the performance using the 3-gram baseline, learned from the whole 487 647 sequences.

**Evaluation.** The evaluation measure was based on perplexity for unseen examples. Given a test set $TS$, it was given by the formula:

$$Score(C, TS) = 2^{-\sum_{x \in TS} Pr_T(x) * log(Pr_C(x))}$$

where $Pr_T(x)$ is the normalized probability of $x$ in the target and $Pr_C(x)$ is the normalized candidate probability for $x$ submitted by the participant. A consequence of this normalization was that adding probability to one of the test strings removed probability from the others. Therefore, this perplexity score measured how well the differences in the assigned probabilities matched with the target probabilities. Notice that this measure is equivalent to the well-known Kullback-Leibler (KL) divergence (Kullback and Leibler, 1951) up to a monotonous transformation.

To decide the final overall rank of each participant, points were attributed for each data set: the leader of a problem at the end of the competition scored 5 points, the second 3, the third 2 and the fourth 1. In case of equality on a problem (based on the 10 first digits of the perplexity score), the earliest submission won. The winner is the participant whose score was the highest. There was no restriction on the number of submissions a given participant could make, but (s)he received no feedback on the resulting score. To compute the final score of a participant, we only considered the best of his submissions to each problem.

## 3. Results

**Competition activity.** 38 participants registered to have access to the problem sets and 16 of them submitted at least one of their solutions to a problem. There were a total number of 2 787 submissions during the competition. 5 participants managed to score some points, 4 of them were ranked first at least once (see Figure 1 in appendix).

During the competition phase, the website received 724 visits (with a maximum of 54 the last day of the competition) from 196 unique visitors with an average visit duration of a bit more than 5 minutes. IPs from 37 countries have been detected, between which 14 countries corresponded to 5 or more visitors.

**Overall results.** The final scores can be seen in Figure 1 and detailed results are presented in table 1. There is a clear winner of PAutomaC: team Shibata Yoshinaka. Of all participants, they obtained the best perplexity values on most instances and performed good on all others. The difference between the perplexity values of the solutions and their submissions was never greater than 0.1. Furthermore, this difference was even smaller on the instances with 100 000 strings, indicating that they make good use of additional data.

Team Shibata Yoshinaka is only outperformed on the (nearly) determinstic ones (Dpfa or Pfa or Hmm with a small transition sparsity). On these instances team Llorens perfoms slightly better. Team Hulden's method also manages to obtain the best perplexity values on two instances, and in fact beat team Llorens overall performance by just 2 points (see the website). Their method seems to perform best on dense instances with few states. The methods used by team Bailly and team Kepler have some difficulty with very sparse instances (and thus also with Dpfa), and perform good but not best on the other instances.

Interestingly, the winning contribution of team Shibata Yoshinaka did not manage to score points on the two real-world problems. Teams Llorens, Hulden, and Kepler, did score points, and outperformed the 3-gram baseline method on these instances. This is not trivial since the solution was biased towards the baseline algorithm as it was also used to provide the probabilities in the solution.

## 4. Conclusion

The results of PAutomaC presented in this paper indicate that the competition was fruitful: refined methods for learning string distributions have been designed and a detailed comparison of their performances is available. In addition, the observation that team Llorens outperforms the winning team on the deterministic instances is very interesting for future research as it could provide a method for deciding whether a given data sample is drawn from a deterministic distribution or from a non-deterministic one, which can be very useful during the discretization of data.

The disclosure of the content of each method will be a very interesting moment and will certainly yield a deeper understanding of string distribution learning algorithms.

## Acknowledgments

We are very thankful to the members of the scientific committee for their help in designing this competition.

## References


L. E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities*, 3:1–8, 1972.

E. Brill, R. Florian, J. C. Henderson, and L. Mangu. Beyond n-grams: Can linguistic sophistication improve language modeling. In *In Proc. of COLING-ACL-98*, pages 186–190, 1998.

R. C. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *Proc. of ICGI'94*, pages 139–150, 1994.

F. Denis, Y. Esposito, and A. Habrard. Learning rational stochastic languages. In *Proc. of* Colt *2006*, volume 4005 of LNCS, pages 274–288. Springer-Verlag, 2006.

Y. Esposito, A. Lemay, F. Denis, and P. Dupont. Learning probabilistic residual finite state automata. In *Proc. ICGI'02*, pages 77–91, 2002.

R. Gavaldà, P. W. Keller, J. Pineau, and D. Precup. Pac-learning of markov models with hidden state. In *Proc. of ECML'06*, pages 150–161, 2006.

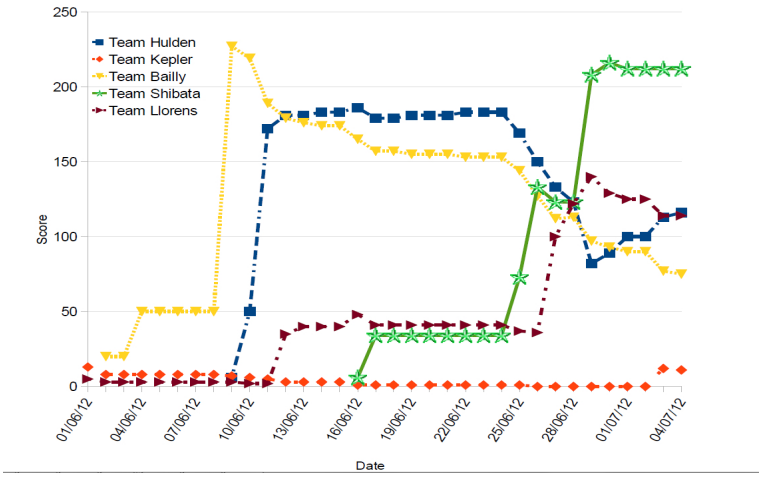
S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1): 79–86, 1951.

A. Van den Bosch, G.J. Busser, W. Daelemans, and S Canisius. An efficient memory-based morphosyntactic tagger and parser for dutch. In *Selected Papers of the 17th Computational Linguistics in the Netherlands Meeting*, pages 99–114, 2007.

S. Verwer, M. de Weerdt, and C. Witteveen. Learning driving behavior by timed syntactic pattern recognition. In *IJCAI'11*, pages 1529–1534, 2011.

S. Verwer, R. Eyraud, and C. de la Higuera. Pautomac: a pfa/hmm learning competition. Technical Report to appear, Radboud University Nijmegen, September 2012. URL www.lif.univ-mrs.fr/~reyraud/PAutomaC.pdf.


## Appendix A. Detailed results.




%stepcounterpage
Figure 1: Overall evolution of the score of the 5 leading teams (artificial data sets).

| Nb | $S$ | $A$ | $S_A$ | $S_T$ | size | type | solution | Hulden | Kepler | Bailly | Shibata | Llorens |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 63 | 8 | 33 | 9 | 20k | Hmm | 29.898 | 30.131 | 30.547 | 30.147 | **29.994** | 30.395 |
| 2 | 63 | 18 | 33 | 2 | 20k | Hmm | 168.331 | 168.455 | 174.866 | 168.429 | 168.430 | **168.420** |
| 3 | 25 | 4 | 79 | 8 | 20k | Pfa | 49.956 | 50.044 | 55.540 | 50.174 | **50.042** | 50.675 |
| 4 | 12 | 4 | 44 | 15 | 100k | Pfa | 80.818 | 80.837 | 85.480 | 80.844 | **80.827** | 80.843 |
| 5 | 56 | 6 | 29 | 2 | 20k | Hmm | 33.235 | 33.241 | 33.427 | 33.237 | **33.237** | 33.238 |
| 6 | 19 | 6 | 48 | 5 | 20k | Dpfa | 66.985 | 67.044 | 82.35 | 67.059 | 67.007 | **67.000** |
| 7 | 12 | 13 | 24 | 8 | 20k | Dpfa | 51.224 | 51.265 | 52.092 | 51.264 | **51.249** | 51.259 |
| 8 | 49 | 8 | 36 | 6 | 100k | Pfa | 81.375 | 81.710 | 85.849 | 81.799 | **81.403** | 81.710 |
| 9 | 71 | 4 | 39 | 1 | 20k | Dpfa | 20.840 | 20.889 | 26.920 | 25.229 | 20.856 | **20.850** |
| 10 | 49 | 11 | 63 | 2 | 20k | Pfa | 33.303 | 33.401 | 34.554 | 33.724 | **33.334** | 34.039 |
| 11 | 47 | 20 | 49 | 2 | 20k | Dpfa | 31.811 | 32.138 | 33.248 | 32.138 | **31.853** | 32.546 |
| 12 | 12 | 13 | 35 | 11 | 20k | Pfa | 21.655 | 21.671 | 21.912 | 21.671 | **21.663** | 21.769 |
| 13 | 63 | 4 | 69 | 2 | 100k | Dpfa | 62.806 | 63.073 | 120.565 | 100.681 | 62.820 | **62.816** |
| 14 | 15 | 12 | 49 | 8 | 20k | Hmm | 116.792 | 116.841 | 118.602 | 116.914 | **116.836** | 116.839 |
| 15 | 26 | 14 | 41 | 7 | 20k | Pfa | 44.242 | 44.285 | 45.208 | 45.285 | **44.274** | 44.701 |
| 16 | 49 | 10 | 62 | 2 | 100k | Dpfa | 30.711 | 30.844 | 31.809 | 35.586 | 30.7187 | **30.7186** |
| 17 | 22 | 13 | 22 | 17 | 20k | Pfa | 47.311 | 47.354 | 48.109 | 48.735 | **47.352** | 47.9215 |
| 18 | 25 | 20 | 23 | 4 | 100k | Dpfa | 57.329 | 57.339 | 57.534 | 76.103 | **57.3316** | 57.3320 |
| 19 | 68 | 7 | 33 | 4 | 100k | Hmm | 17.877 | 17.930 | 18.816 | 19.316 | **17.880** | 17.919 |
| 20 | 11 | 18 | 39 | 15 | 20k | Hmm | 90.972 | 91.016 | 95.304 | 91.351 | **90.999** | 93.504 |
| 21 | 56 | 23 | 25 | 5 | 20k | Hmm | 30.519 | 30.605 | 35.578 | 30.714 | **30.568** | 32.217 |
| 22 | 55 | 21 | 25 | 6 | 100k | Pfa | 25.982 | 26.078 | 26.136 | 26.010 | **25.988** | 26.080 |
| 23 | 33 | 7 | 38 | 11 | 100k | Hmm | 18.408 | 18.418 | 18.720 | 18.547 | **18.413** | 18.447 |
| 24 | 6 | 5 | 50 | 17 | 20k | Dpfa | 38.729 | 38.737 | 42.366 | 38.753 | **38.7317** | 38.7322 |
| 25 | 40 | 10 | 58 | 5 | 20k | Hmm | 65.735 | 65.978 | 67.929 | 66.069 | **65.783** | 67.266 |
| 26 | 73 | 6 | 59 | 1 | 20k | Dpfa | 80.743 | 82.657 | 111.502 | 141.082 | **80.833** | 80.837 |
| 27 | 19 | 17 | 64 | 5 | 20k | Dpfa | 42.427 | 42.473 | 43.511 | 42.712 | 42.464 | **42.456** |
| 28 | 23 | 6 | 75 | 11 | 20k | Hmm | 52.744 | 52.855 | 53.583 | 53.084 | **52.841** | 53.198 |
| 29 | 36 | 6 | 38 | 4 | 20k | Pfa | 24.031 | 24.199 | 28.580 | 24.817 | **24.042** | 24.106 |
| 30 | 9 | 10 | 66 | 18 | 20k | Pfa | 22.926 | **22.932** | 23.394 | 22.960 | 22.934 | 23.211 |
| 31 | 12 | 5 | 38 | 20 | 20k | Pfa | 41.214 | 41.243 | 42.531 | 41.417 | **41.233** | 41.623 |
| 32 | 43 | 4 | 77 | 2 | 100k | Dpfa | 32.613 | 32.743 | 41.975 | 38.300 | 32.622 | **32.619** |
| 33 | 13 | 15 | 59 | 12 | 20k | Hmm | 31.865 | 31.872 | 32.2194 | 31.920 | **31.871** | 32.030 |
| 34 | 64 | 21 | 37 | 3 | 20k | Pfa | 19.955 | 20.428 | 20.581 | 20.476 | **19.969** | 20.542 |
| 35 | 47 | 20 | 36 | 2 | 20k | Dpfa | 33.777 | 34.326 | 34.714 | 33.835 | **33.800** | 34.295 |
| 36 | 54 | 9 | 63 | 7 | 100k | Hmm | 37.986 | 38.203 | 38.206 | 38.176 | **38.018** | 38.405 |
| 37 | 69 | 8 | 52 | 18 | 100k | Pfa | 20.980 | 21.016 | 21.025 | 21.027 | **21.001** | 21.016 |
| 38 | 14 | 10 | 79 | 19 | 20k | Hmm | 21.446 | 21.494 | 21.650 | 21.514 | **21.459** | 21.596 |
| 39 | 6 | 14 | 42 | 18 | 20k | Pfa | 10.002 | **10.0029** | 10.054 | 10.005 | 10.0034 | 10.004 |
| 40 | 65 | 14 | 65 | 2 | 20k | Dpfa | 8.201 | 8.255 | 8.366 | 8.496 | 8.207 | **8.206** |
| 41 | 54 | 7 | 69 | 14 | 100k | Hmm | 13.912 | 13.941 | 13.942 | 13.932 | **13.921** | 13.940 |
| 42 | 6 | 9 | 52 | 17 | 20k | Dpfa | 16.004 | 16.008 | 16.080 | 16.008 | 16.007 | **16.005** |
| 43 | 67 | 5 | 60 | 16 | 20k | Pfa | 32.637 | 32.747 | 32.841 | 32.817 | **32.723** | 32.777 |
| 44 | 73 | 13 | 63 | 6 | 20k | Hmm | 11.709 | 11.798 | 11.920 | 11.778 | **11.725** | 12.041 |
| 45 | 14 | 19 | 80 | 9 | 20k | Hmm | 24.042 | 24.048 | 24.252 | 24.084 | 24.050 | **24.045** |
| 46 | 19 | 23 | 49 | 10 | 20k | Pfa | 11.982 | 11.999 | 12.136 | 12.082 | **11.988** | 12.106 |
| 47 | 61 | 15 | 30 | 2 | 100k | Dpfa | 4.1190 | 4.124 | 4.144 | 4.120 | 4.1192 | **4.1191** |
| 48 | 16 | 23 | 70 | 6 | 20k | Dpfa | 8.036 | 8.042 | 8.183 | 8.045 | **8.039** | 8.191 |
| r1 | | 11 | | | 10k | 3gram | 70.112 | **70.581** | 70.659 | 79.570 | 71.083 | 70.599 |
| r2 | | 18 | | | 20k | 3gram | 5.108 | 5.126 | 5.128 | | 5.129 | **5.124** |

Table 1: Perplexity scores of active participants and the solutions for all problem instances, along with their parameters: number of states ($S$), alphabet size ($A$), symbol sparsity ($S_A$), transition sparsity ($S_T$), size of training set, and type of machine.